```
#-------------------------------------------- Part-1 Understanding the code --------------------------------------------------------


# Importing necessary modules from the sklearn library for the project

from sklearn import datasets
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier


# Loading and Preparing the Iris Dataset
# This dataset contains measurements of iris flowers in different categories

iris = datasets.load_iris()
data, labels = iris.data, iris.target



# Split the dataset into training and testing sets using an 80-20 ratio
# 'train_test_split' automatically shuffles the data and splits it into train and test

res = train_test_split(data, labels, train_size=0.8,test_size=0.2,random_state=12)
train_data, test_data, train_labels, test_labels = res



knn = KNeighborsClassifier()

# Fit the KNN model on the training data
# The model learns to classify based on the distances between the training data points

knn.fit(train_data, train_labels)
```

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

```
# Use the trained model to predict the labels of the training data

learn_data_predicted = knn.predict(train_data)

# Output the predictions and the true labels of the training data
print("Predictions from the classifier:")
print(learn_data_predicted)

print("Target values:")
print(train_labels)
print('\n')
# Calculate and print the accuracy score on the training set
# Accuracy is the ratio of correct predictions to total prediction

# Assuming learn_data_predicted and train_labels are already defined
accuracy = accuracy_score(learn_data_predicted, train_labels)
print('Accuracy of the Train lables: ' + str(accuracy))
```

```
Predictions from the classifier:
[0 1 2 0 2 0 1 1 0 1 1 0 0 0 0 0 0 0 2 0 2 1 1 1 0 2 1 1 2 0 2 0 2 1 2 2 1
 1 1 2 2 0 2 2 0 1 0 2 2 0 1 1 0 0 1 1 1 1 2 1 2 0 0 1 1 2 0 2 1 0 2 2 1 2
 2 0 0 2 1 1 2 0 1 1 0 1 1 2 2 1 0 2 0 2 0 0 1 2 2 1 2 2 0 1 1 0 2 2 2 1 2
 2 2 0 0 1 0 2 2 1]
Target values:
[0 1 2 0 2 0 1 1 0 1 1 0 0 0 0 0 0 0 2 0 2 1 1 1 0 2 1 1 2 0 2 0 2 2 2 2 1
 1 1 1 2 0 2 2 0 1 0 2 2 0 1 1 0 0 1 1 1 1 2 1 2 0 0 1 1 1 0 2 1 0 2 2 1 2
 2 0 0 2 1 1 2 0 1 1 0 1 1 2 2 1 0 2 0 2 0 0 1 2 2 1 2 2 0 1 1 0 2 2 2 1 2
 2 2 0 0 1 0 2 2 1]


Accuracy of the Train lables: 0.975
```

```
# Re-create the KNN classifier, this time with specific parameters
knn2 = KNeighborsClassifier(algorithm='auto',  leaf_size=30, metric='minkowski',p=2, n_neighbors=5,  weights='uniform')

# Fit the model to the training data again
knn2.fit(train_data, train_labels)
```

```
test_data_predicted = knn2.predict(test_data)

# Calculate and print the accuracy score on the test data
accuracy = (accuracy_score(test_data_predicted, test_labels))
print('Accuracy of the test lables :' + str(accuracy))
```

```
Accuracy of the test lables :0.9666666666666667
```

```
#============================================  Part-2 Working with a Simulated Dataset ====================================================
```

```
# Importing necessary libraries
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
import pandas as pd


#Create a simulated dataset using make_blobs
# Define the centers of the simulated blobs (i.e., clusters)

centers = [[2, 4], [6, 6], [1, 9]]
data, labels = make_blobs(n_samples=150, centers=np.array(centers), random_state=1)


# Convert to DataFrame to include column names
df = pd.DataFrame(data, columns=['Feature 1', 'Feature 2'])
df['Label'] = labels

# Print the first 5 rows of the simulated dataset
print("First 5 rows of the simulated dataset:")
print(df.head())
```

```
First 5 rows of the simulated dataset:
   Feature 1  Feature 2  Label
0  -0.236853   9.875839      2
1   1.016528   9.177188      2
2   2.558806   9.109403      2
3   5.863555   5.880946      1
4   7.121418   6.408901      1
```

```
# Split the simulated dataset into training (80%) and testing (20%)
train_data, test_data, train_labels, test_labels = train_test_split(data, labels, train_size=0.8, test_size=0.2, random_state=12)



# Create a KNN classifier
knn_simulated = KNeighborsClassifier(n_neighbors=5, algorithm='auto', metric='minkowski', p=2)
knn_simulated.fit(train_data, train_labels)
```

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

```
# Predict the train data labels
train_data_predicted_simulated = knn_simulated.predict(train_data)


# Print predictions and target values for the training set
print("\nTraining Set Predictions from the classifier:")
print(train_data_predicted_simulated)
print("Training Set Target values:")
print(train_labels)
```

```
Training Set Predictions from the classifier:
[0 2 1 0 0 1 1 2 2 0 2 2 2 1 1 0 0 2 1 1 0 0 0 1 1 2 0 0 1 0 1 1 1 0 1 2 0
 1 0 1 2 2 2 0 2 0 2 2 0 0 0 1 2 2 2 2 1 1 0 1 2 1 2 2 2 0 0 0 0 0 0 0 1 1
 2 1 2 1 2 2 1 1 1 0 2 1 2 1 0 1 2 1 0 2 0 1 2 2 0 2 1 0 0 2 1 1 2 2 0 1 1
 1 2 2 2 1 1 2 1 2]
Training Set Target values:
[0 2 1 0 0 1 1 2 2 0 2 2 2 1 1 0 0 2 1 1 0 0 0 1 1 2 0 0 1 0 1 1 1 0 1 2 0
 1 0 1 2 2 2 0 2 0 2 2 0 0 0 1 2 2 2 2 1 1 0 1 2 1 2 2 2 0 0 0 0 0 0 0 1 1
 2 1 2 1 2 2 1 1 1 0 2 1 2 1 0 1 2 1 0 2 0 1 2 2 0 2 1 0 0 2 1 1 2 2 0 1 1
```

```
 1 2 2 2 1 1 2 1 2]
```

```
# Calculate and print accuracy for the simulated training data
train_accuracy_simulated = accuracy_score(train_data_predicted_simulated, train_labels)
print("Training Accuracy for simulated dataset: " + str(train_accuracy_simulated))
```

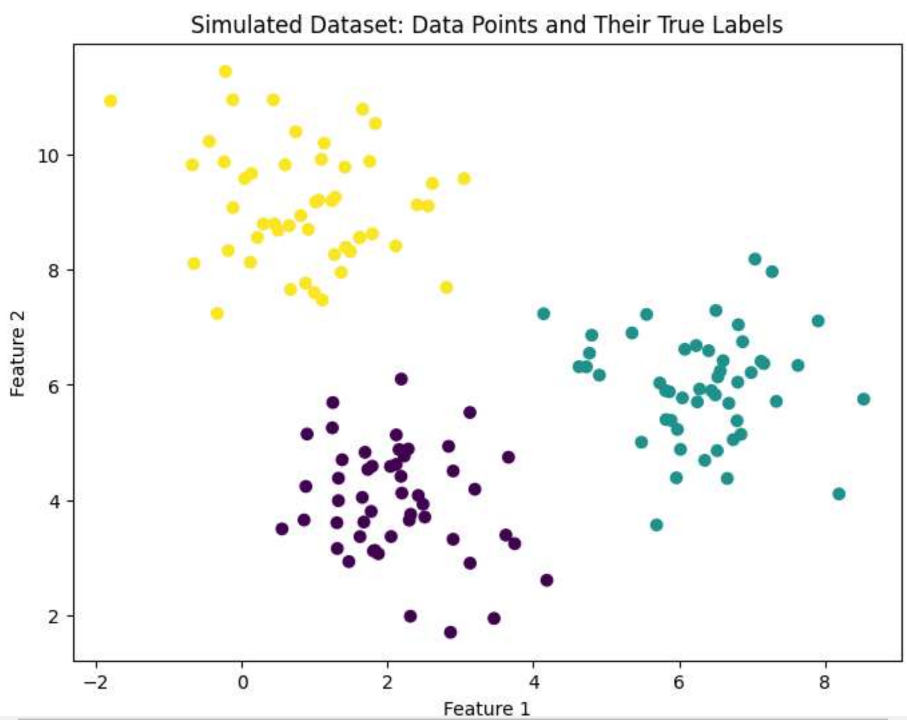
```
Training Accuracy for simulated dataset: 1.0
```

```
# Predict the test data labels
test_data_predicted_simulated = knn_simulated.predict(test_data)

# Calculate and print accuracy for the simulated test data
test_accuracy_simulated = accuracy_score(test_data_predicted_simulated, test_labels)
print("Test Accuracy for simulated dataset: "+ str(test_accuracy_simulated))
```

```
Test Accuracy for simulated dataset: 1.0
```

```
# Plot the simulated dataset
plt.figure(figsize=(8, 6))
plt.scatter(data[:, 0], data[:, 1], c=labels, cmap='viridis')
plt.title('Simulated Dataset: Data Points and Their True Labels')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```




```
# Create a mesh grid for the feature space
x_min, x_max = data[:, 0].min() - 1, data[:, 0].max() + 1
y_min, y_max = data[:, 1].min() - 1, data[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
                     np.arange(y_min, y_max, 0.1))

# Predict class labels across the entire feature space
Z = knn_simulated.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Plot decision boundaries and overlay the training and test data
plt.figure(figsize=(8, 6))
plt.contourf(xx, yy, Z, alpha=0.8, cmap='viridis')
plt.scatter(test_data[:, 0], test_data[:, 1], c=test_labels, edgecolor='k', s=100, label='Test Data')
plt.scatter(train_data[:, 0], train_data[:, 1], c=train_labels, edgecolor='k', marker='X', s=40, label='Train Data')
```

```
plt.title("Decision Boundaries and Data Points for Simulated Dataset")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.legend()
plt.show()
```